REMARKS

Claims 1-39 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 102(e) Rejection:

The Office Action rejected claims 1-39 under 35 U.S.C. § 102(e) as being anticipated by Sameer, et al. ("Core Java Data Objects: Chapter 3 "Getting Started with JDO"") (hereinafter "Sameer"). Applicant traverses the rejection for at least the following reasons.

As an initial matter, the 102(e) rejection of claims 1-39 as being anticipated by Sameer is improper because Sameer does not qualify as prior art under 102(e). More specifically, 35 U.S.C. § 102 recites in part:

> A person shall be entitled to a patent unless - (e) the invention was described in - (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for the purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language. (emphasis added)

Sameer is neither a patent nor an application for patent according to the requirement of 35 U.S.C. § 102(e). Instead, Sameer is non-patent literature. Accordingly, the 102(e) rejection of claims 1-39 as being anticipated by Sameer is improper and removal thereof is respectfully requested.

Claim 1

In regard to claim 1, the Examiner has not clearly specified the particular

portion of the cited art on which she is relying to teach the claimed "persistence structure." For instance, the Examiner cites page 53, lines 14-22 in regard to the determination of a persistence structure according to claim 1; however, the Examiner's citation does not make it unambiguously clear as to the specific element of Sameer that she considers to be equivalent to the "persistence structure" of Applicant's claim. **Applicants note that per MPEP 707.07(d), the ground of rejection in an Examiner's Action should be "fully and clearly stated." Furthermore, 37 CFR 1.104(c)(2) states that "[w]hen a reference is complex or shows or describes inventions other than that claimed by the applicant, the particular part relied on must be designated as nearly as practicable."** Since the current rejection does not unambiguously specify the particular element of Sameer relied upon to allegedly teach the claimed "persistence structure," Applicants assert that the Examiner's rejection is not "fully and clearly stated" nor has the Examiner designated the particular part(s) relied on "as nearly as practicable." Accordingly, Applicants request that the Examiner specify the **particular** element of Sameer on which she is relying to teach the claimed "persistence structure." Based on the content of the instant Office Action, the Applicant simply cannot ascertain the basis for the Examiner's rejection in regard to this claim limitation. Accordingly, a *prim facie* rejection has not been stated.

Furthermore, Sameer fails to teach a class structure based data object enhancer configured to analyze the structure of the one or more classes to determine a persistence structure specifying data fields of the one or more classes to be persisted. The Examiner cites page 53, lines 14-22 of Sameer, which are reproduced below.

> When the Author source and metadata files are ready, they can be passed to the byte code enhancement tool. This takes care of implementing the persistence-capable contract. As an example, if using the JDO reference implementation (RI), the Author class would be compiled as normal using javac and then the RI byte code enhancement tool would be invoked from the command line as follows:
>
> javac Author.java
> java com.sun.jdori.enhancer.Main Author.class Author.jdo

> By default, the RI enhancement tool updates the Author byte code in place, and the original Author.class file is replaced with an enhanced one.

Sameer teaches "enhancing" a class file by using a byte code enhancement tool. The inputs to the byte code enhancement tool are source information for a class file (the "Author source" mentioned above) and associated metadata, neither of which include a persistence structure according to the specific limitations of claim 1. In fact, Sameer is silent with respect to "a persistence structure specifying data fields of the one or more classes to be persisted," much less determining such a persistence structure by "analyz[ing] the structure of the one or more classes." Sameer provides additional details about the manner in which persistence capable classes are created on page 49, such as by utilizing source code generation, source code preprocessing, and byte code enhancement. Neither these portions of Sameer nor the portions cited by the Examiner teach "a persistence structure specifying data fields of the one or more classes to be persisted," much less determining such a persistence structure by "analyz[ing] the structure of the one or more classes." Sameer simply does not teach the specific limitations of Applicant's claim.

Furthermore, Sameer fails to teach a class structure based data object enhancer configured to generate one or more enhanced classes corresponding to the one or more classes such that an object of the one or more enhanced classes is enhanced to persist data of the data fields to be persisted according to the persistence structure, wherein said data of the data fields to be persisted is data of said object; wherein the generation of each of said one or more enhanced classes comprises adding to the corresponding one of said one or more classes, one or more calls to persist data fields as specified by the persistence structure. Sameer fails to mention anything at all about generating enhanced classes by "adding to the corresponding one of said one or more classes, one or more calls to persist data fields as specified by the persistence structure." In fact, as demonstrated above, Sameer fails to explicitly disclose a persistence structure, much less adding "one or more calls to persist data fields as specified by the persistence structure." On page 3 of the Office Action of June 25, 2009, the Examiner cites the "byte code enhancement process" of page 49 of

Sameer as well as Figure 3-2 of Sameer. In these portions of the reference, Sameer describes the manner in which a "byte code enhancement tool" "either updates the existing class files or creates new ones." Nothing about the operation of the "byte code enhancement tool" of Sameer includes generating enhanced classes by "adding to the corresponding one of said one or more classes, one or more calls to persist data fields as specified by the persistence structure." This portion of Applicant's claim is simply not taught by the cited reference.

"Unless a reference discloses within the four corners of the document not only all of the limitations claimed but also all of the limitations arranged or combined in the same way as recited in the claim, it cannot be said to prove prior invention of the thing claimed and, thus, cannot anticipate under 35 U.S.C. § 102." *Net MoneyIN, Inc. v. VeriSign et al.*, Case No. 07-1565 (Fed. Cir., Oct. 20, 2008). Since Sameer fails to teach all of the limitations of claim 1, Sameer cannot be said to anticipate claim 1.

Thus, for at least the reasons presented above, the rejection of claim 1 is unsupported by the teachings of the cited art, and removal thereof is respectfully requested. Similar remarks apply to claims 14 and 27.

Claim 2

In regard to claim 2, Sameer fails to teach wherein to analyze the structure of the one or more classes, the class structure based enhancer is configured to make one or more Java reflection calls to the one or more classes. The Examiner cites various portions of Sameer, including mapping a class to a data store (*see e.g.,* section 3.3.2 of pages 53-54 of Sameer). None of the cited portions of Sameer mention anything at all about Java reflection calls, much less "wherein to analyze the structure of the one or more classes, the class structure based enhancer is configured to make one or more Java reflection calls to the one or more classes" according to the specific limitation of claim 2.

The Examiner asserts that "it is inherent of using Java reflection for mapping the fields of the persistent objects." However, "in relying upon the theory of inherency, the examiner must provide a basis in fact and/or technical reasoning to reasonably support the determination that the allegedly inherent characteristic necessarily flows from the teachings of the applied prior art." *Ex parte Levy*, 17 USPQ2d 1461, 1464 (Bd. Pat. App. & Inter. 1990) (emphasis in original). One of ordinary skill in the art would recognize that the mapping of a class to a data store could be performed in any number of ways that do not include utilizing Java reflection calls; for instance, an administrator could create a configuration file that specifies such mapping. Accordingly, the Examiner's assertion that "it is inherent of using Java reflection for mapping the fields of the persistent objects" does not necessarily flow from the teachings of the cited art.

Furthermore, Sameer explicitly teaches that the implementation of the "mapping" on which the Examiner relies is left up to the developer. Accordingly, by definition such "mapping" does not necessarily include utilizing Java reflection calls. For instance, in section 3.3.2, Sameer teaches "JDO does not define how to specify how the fields of a class should be mapped to the underlying data store" and "all this is datastore and JDO-implementation specific … a JDO implementation provides the necessary tools to allow a developer to define a mapping between the persistence capable classes and the underlying data store." There is no reason why the developer would necessarily have to utilize Java reflection calls to perform such mapping. Again, as on example, a developer could create a configuration file that specifies such mapping without using Java reflection calls.

As demonstrated above, Sameer fails to mention anything at all about Java reflection calls, and the Examiner's reliance on inherency to teach the limitations of Applicant's claim is clearly improper. Accordingly, Sameer cannot be relied upon to teach wherein to analyze the structure of the one or more classes, the class structure based enhancer is configured to make one or more Java reflection calls to the one or more classes.

Thus, for at least the reasons presented above, the rejection of claim 2 is unsupported by the cited art and removal thereof is respectfully requested. Similar remarks apply to claim 15 and 28.

Section 102(b) Rejection:

The Office Action rejected claims 1-39 under 35 U.S.C. § 102(b) as being anticipated by Lipkin et al. (U.S. Publication 2002/0120859) (hereinafter "Lipkin"). Applicant traverses the rejection for at least the following reasons.

Claim 1

In regard to claim 1, the Examiner has not clearly specified the particular portions of the cited art on which she is relying to teach the claimed "one or more classes," "persistence structure" and "one or more enhanced classes." The Examiner's broad citation of various paragraphs of Lipkin does not make it unambiguously clear as to the specific elements of Lipkin that she considers to be equivalent to the claimed "one or more classes," "persistence structure" and "one or more enhanced classes" of Applicant's claim. **Applicants note that per MPEP 707.07(d), the ground of rejection in an Examiner's Action should be "fully and clearly stated." Furthermore, 37 CFR 1.104(c)(2) states that "[w]hen a reference is complex or shows or describes inventions other than that claimed by the applicant, the particular part relied on must be designated as nearly as practicable."** Since the current rejection does not unambiguously specify the particular elements of Lipkin relied upon to allegedly teach the claimed the claimed "one or more classes," "persistence structure" and "one or more enhanced classes," Applicants assert that the Examiner's rejection is not "fully and clearly stated" nor has the Examiner designated the particular part(s) relied on "as nearly as practicable." Accordingly, Applicants request that the Examiner specify the **particular** elements of Lipkin on which she is relying to teach the claimed "one or more classes," "persistence structure" and "one or more enhanced classes." Based on the content of the instant Office Action, the Applicant simply cannot

ascertain the basis for the Examiner's rejection in regard to these claim limitations. Accordingly, a *prim facie* rejection has not been stated.

In regard to claim 1, Lipkin does not teach a class structure based data object enhancer configured to analyze the structure of the one or more classes to determine a persistence structure specifying data fields of the one or more classes to be persisted, and generate one or more enhanced classes corresponding to the one or more classes such that an object of the one or more enhanced classes is enhanced to persist data of the data fields to be persisted according to the persistence structure. The Examiner cites paragraphs [0253]-[0295] of Lipkin. According to the teachings of Lipkin, there is no analysis of one or more classes to determine a persistence structure and there is no generation of one or more enhanced classes that correspond to the one or more classes. Instead, Lipkin teaches a system where all classes are derived from a single base class that already includes mechanisms for persistence (e.g., "save, restore, and delete capabilities"; see paragraph [0054] of Lipkin below). Such classes are derived via inheritance as subclasses of a common base class. Lipkin teaches this concept in paragraph [0254], which is reproduced below.

> In a preferred embodiment all business objects that Saba's Application server manipulates are derived from a single base class called SabaObject. The SabaObject class provides save, restore, and delete capabilities by implementing the persistence layer architecture. All subclasses of Saba Object then inherit this behavior and rarely if ever override it.

Presumably the Examiner considers the subclasses of SabaObject to be equivalent to the "one or more enhanced classes" of Applicant's claim. However, SabaObject subclasses are not generated in the manner according to which the "one or more enhanced classes" of claim 1 are generated. For instance, the generation of a SabaObject subclass does not include analyzing one or more classes to determine a persistence structure and generating a SabaObject subclass such that an object of the SabaObject subclass is enhanced to persist data of the data fields to be persisted according to the persistence structure. Such concept is not in anyway disclosed by Lipkin. Instead, Lipkin teaches that SabaObject subclasses are generated via inheritance. One of ordinary skill in the art would immediately recognize that inheritance is a basic tenet of object oriented

programming that does not include generating one or more enhanced classes based on a persistence structure.

Furthermore, Lipkin fails to teach wherein the generation of each of said one or more enhanced classes comprises adding to the corresponding one of said one or more classes, one or more calls to persist data fields as specified by the persistence structure. The Examiner cites paragraphs 0231, 0235-0237, 0248, 0284, and 0286 of Lipkin, none of which teaches the specific limitations of Applicant's claim. More specifically, the cited portions of Lipkin are silent with respect to adding to one or more classes, one or more calls to persist data fields as specified by a persistence structure. This limitation of Applicant's claim is simply not taught by the cited reference. Instead, as described above, all SabaObject subclasses already include mechanisms for persistence that are inherited from a common base class. Such inheritance does not include adding one or more calls to classes in the specific manner recited by claim 1. Furthermore, Lipkin does describe a process by which a user may provide information about customer fields (see e.g., paragraph [0284] of Lipkin). However, the actual classes of Lipkin are not changed in response to such information, nor are one or more calls added to classes as specified by claim 1. Instead, to accommodate custom fields, Lipkin's algorithms for save and restore includes additional steps (*see e.g.,* paragraphs [0286]-[0292]). Such algorithms are not part of Lipkin's SabaObject classes and thus this portion of Lipkin cannot be considered commensurate with adding to one or more classes, one or more calls to persist data fields as specified by the persistence structure. Configuring algorithms in a certain manner is not commensurate with adding one or more persistence-based calls to one or more classes.

Thus, for at least the reasons presented above, the rejection of claim 1 is unsupported by the teachings of the cited art, and removal thereof is respectfully requested. Similar remarks apply to claims 14 and 27.

In regard to the 102(e) and 102(b) rejections, Applicants also assert that numerous ones of the dependent claims recite further distinctions over the cited art. However, since the rejection has been shown to be unsupported for the independent claims, a further discussion of the dependent claims is not necessary at this time. Applicants reserve the right to present additional arguments.

CONCLUSION

Applicant submits the application is in condition for allowance, and an early notice to that effect is respectfully requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5681-72300/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicant

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: September 25, 2009